Cambridge Scientific Center

36. Y05
June 1966

IBM
Data Processing Division

String Processing on the System/360: Techniques and Example

String Processing on the System/360: Techniques and Example

S. E. Madnick
IBM Cambridge Scientific Center Report

International Business Machines Corporation
Cambridge Scientific Center
Cambridge, Massachusetts
June, 1966

36.Y05
June, 1966
Scientific Center Report
Limited Distribution

STRING PROCESSING ON THE SYSTEM/360: TECHNIQUES AND EXAMPLE

S. E. Madnick

International Business
Machines Corporation
Cambridge, Massachusetts

Abstract

The IBM System/360 Data Processing System has many capabilities that facilitate the implementation of a variety of efficient string processing techniques. This paper presents six of these techniques with an example of the successful use of one of them.

Index Terms for the IBM Subject Index

IBM 0360-40
Programs
Data Processing
Artificial Intelligence
Information Processing Language
05-Computer Application
21-Programming

LIMITED DISTRIBUTION NOTICE

This report has been submitted for publication elsewhere and has been issued as a Technical Report for early dissemination of its contents. As a courtesy to the intended publisher, it should not be widely distributed until after the date of outside publication.

TABLE OF CONTENTS

I. Introduction

The title of this paper was also that of a discussion group held at the March 29 ACM Symposium on Symbolic and Algebraic Manipulation. At that time it became obvious to the writer of this paper that many people are interested in developing string processing languages or utilizing string processing techniques in the solution of problems. Although there is a reasonable amount of documentation describing the external appearances of many existing string processing languages, there is a noticeable lack of information about their internal organization.

The System/360 has many capabilities that facilitate the implementation of a variety of string processing techniques. This paper presents six of these techniques along with an example of the successful use of one of them. Of course, many variations of the presented techniques are possible.

In general three basic operations are performed in string processing: (1) creation of strings; (2) examination of strings; and (3) alteration of strings. The techniques described in this paper are presented from the point of view of these three basic operations. Speed of operation, storage requirements, and programmer convenience are also considered.

II. The IBM System/360

Familiarity with a number of System/360 capabilities is essential to an understanding of the string processing techniques presented in this paper. For this reason a brief discussion of the relevant points has been included. For more detailed information consult the IBM System/360: "Principles of Operation" manual.

The System/360 has sixteen 32-bit general purpose registers which can be used as accumulators, index registers, or base registers. The basic addressable memory element is 8 bits long and is called the byte. For performing fixed length operations, as on the IBM 7094, four bytes can be handled as a single 32-bit word. For performing variable-length operations, as on the IBM 1401 and 1620, an arbitrary number of bytes can be moved and compared storage-to-storage without using any of the general purpose registers.

The effective address of an instruction is a 24-bit number formed by the sum of (1) the 12-bit displacement, which is part of the instruction, (2) the 24 low order bits of the base register specified by the instruction, and (3) the 24 low order bits of the index register, also specified by the instruction.

It is important to note the storage to storage capability and the equivalence of accumulators, index registers and base registers.

III. The Problem of String Processing

A Symbol (character, letter, digit) on the System/360 is represented by an 8-bit code and thus, in size, corresponds to the byte. The simplest way to store strings would be to put consecutive characters in successive bytes throughout memory. However, any attempt to change the length of the string results in considerable character-moving.

Most symbol manipulating languages solve this problem by use of pointers. A pointer allows the elements of the string to be locatedin physically noncontiguous regions of the computer's memory and yet be logically bound together.

A pointer on the System/360 must be 24-bits long to connect string sections which are located arbitrarily in the computer. There are several symbol - pointer arrangements possible with the System/360.

IV. Description of Data Structures Under Consideration

Six basically different data structures with potential for numerous variations have been devised. They are described below and schematically presented in Figure 1. The first four methods are based primarily upon fixed word length considerations, while the remaining two methods make use of the variable word length features.

Method 1 (Double-word Blocks):

The string is represented internally by linked two-word blocks. The first word contains a character, the second contains a pointer to the next

character.

Method 2 (Single-word Blocks):

This method strongly resembles the double-word block technique, but, rather than using two words, the 8-bit character and 24-bit pointer are packed into a single 32-bit word.

Method 3 (Variable Length Blocks):

The characters are stored one to a word consecutively in memory. Whenever the sequence is to be broken, a pointer indicates the location of the next block of characters. Characters and pointer can be identified by information stored in the unused portion of the 32-bit word.

Method 4 (Packed Double-word Blocks):

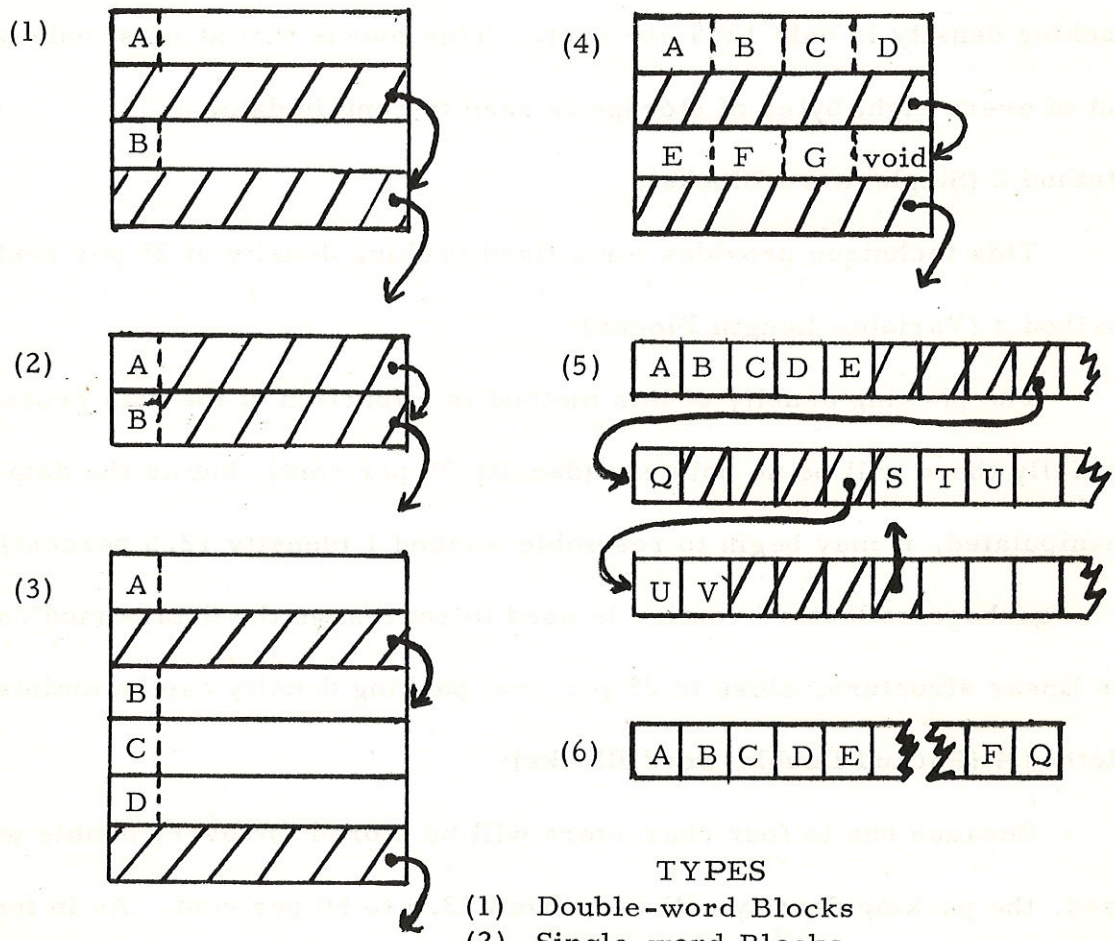
The characters are stored in fixed length packed blocks (4 per word, 8 per double word, etc.) followed by a pointer to the next block. For the example presented in this paper, four characters are stored in a word followed by a pointer to the next four characters. A special character called the "void" character fills the empty spaces in data blocks that are only partially filled.

Method 5 (Linked Linear Strings):

Characters are stored sequentially in memory, byte by byte. Wherever the sequence is to be broken, a special character is used to denote a pointer. In other words, the pointer is made 32-bits long where the leading 8-bits identify it as a pointer.

Method 6 (Linear String):

This method can be implemented in at least two ways. The simplest (conceptually) is always to maintain strings linearly throughout memory without any pointers. An alternate scheme is to store strings linearly within large blocks (4096 characters long for example) with a pointer to the next block.



TYPES

(1) Double-word Blocks
(2) Single-word Blocks
(3) Variable-length Blocks
(4) Packed Double-word Blocks
(5) Linked Linear Strings
(6) Linear String

Figure 1. Data Structures Under Consideration

V. Storage Requirements of Data Structures Under Consideration

In discussing storage requirements the term "packing density" is used. Packing density is the percentage of storage containing character information.

Method 1 (Double-word Blocks):

Since only one character is stored for every double word used, the packing density is only 12.5 per cent. This means that at most only one out of every eight bytes of storage is used to contain data.

Method 2 (Single-word Blocks):

This technique provides for a fixed packing density of 25 per cent.

Method 3 (Variable Length Blocks):

The packing density of this method is a function of the data processed. Initially there will be no pointers (density 25 per cent), but as the data is manipulated, it may begin to resemble method 1 (density 12.5 percent). If a "garbage collector" routine is used to rearrange the data periodically its linear structure, close to 25 per cent packing density can be maintained.

Method 4 (Packed Double-word Blocks):

Because one to four characters will be stored for every double word used, the packing density will vary from 12.5 to 50 per cent. As in method 3, a garbage collector could be used to maintain storage density.

Method 5 (Linked Linear Strings):

Initially all the characters will be stored linearly throughout memory. This results in a packing density of 100 per cent. Under worst case conditions

each character could be followed by a pointer, thus reducing the density to 25 per cent. The use of a garbage collector to reorganize the data periodically can keep the density as close to 100 per cent as desired.

Method 6 (Linear Strings):

The linear storage technique results in an 100 per cent packing density. Of course this method requires continual storage reorganization.

VI. Speed Limitations Posed by Data Structures Under Consideration

The ease with which certain string manipulations can be performed determines, to a large extent, the overall operating speed of a string processing application. The basic string manipulating operations are:

(i) a scan
(ii) an add/delete
(iii) a storage manager or "garbage collector".

Method 1 (Double-word Blocks):

Individual characters can be moved or compared either by using the System/360's storage-to-storage character processing capabilities, or by loading into a general purpose register and performing fixed word length operations on them.

The next element of the string can be easily accessed since the pointer is kept in the low order 24 bits of the pointer word. In this case the pointer is immediately loaded into a data base register.

To delete a character or group of characters from the string, it is necessary merely to change the pointer preceding the portion to be deleted

to point to the first character after the section. To add a section to the string the reverse process is used. (Before a group of characters can be inserted into the string, they must be linked together in the same form as in the string). The pointer located on the string at the place where the insertion is to be made is moved to the bottom of the section to be inserted. It is replaced by a pointer to the first element of the new section.

There are two possible techniques that can be used to maintain free storage from which new strings are formed. One method uses a portion of available memory for stored strings, and the remainder as a bulk quantity of unused storage. A pointer keeps track of the beginning of the free storage area. As new strings are produced, the free storage is reduced. When no free storage remains, the garbage collector must move and relink the string to create free storage from deleted elements.

Another method of maintaining free storage is to place every word of available storage on a string. This special string, called the "free string", links all unused words of storage. When sections are to be added to a regular string, the necessary number of elements is unlinked from the free string. Conversely when a section of a string is deleted, it is added to the free string. No garbage collection is necessary since all free storage is linked together.

There is one more consideration: a multi-programming environment with automatic paging where program segments are swapped between main memory and secondary storage. Effective use of paging requires that the

data being referenced are fairly localized. In general the elements of the strings are located through memory. Since the double-word blocks method provides a pointer for every character, it is possible for each character to be located in a different section of memory. Although variations of this method, that tend to localize the string, have been developed, the complexity involved outweighs the simplicity of the original method.

Method 2 (Single-word Blocks):

This technique has the same basic characteristics as Method 1. Since only the low order 24-bits of the base register are used to determine an address, it is not necessary to mask off the 8 high order bits containing the character. The only difference is a slight additional manipulation involved in the insertion of pointers without destroying the character, into the single word block.

Method 3 (Variable Length Blocks):

The characters can be manipulated by any of the methods described above. The next element of a string can be obtained by incrementing the data base register, if a pointer is not present, or by loading the pointer into the data base register, if the end of a block has been reached. It is necessary continually to check the data to distinguish between characters and pointers.

Deletion of characters is simple. The first character to be deleted is replaced by a pointer to the character following the section to be deleted. Addition to the character string is not quite as easy. The characters to be inserted are put in consecutive words of a block obtained from free storage. The letter located at the spot where the addition is to be made is moved to the top of the new block and replaced by a pointer to its new location. The

last element of the new block is a pointer back to the element of the string immediately following the point of insertion.

The presence of odd sized blocks and the need for a contiguous free storage area make a garbage collector the only practical means of maintaining the storage.

Since the strings are more localized than in method 1 and 2, the variable length block method is more practical for a computing system utilizing paging techniques.

Method 4 (Packed Double-word Blocks):

The characters can be removed from the packed word, byte by byte, or the entire word can be placed in a register and shifted, one character at a time. The "void" character must be detected and ignored. After all four characters have been processed, the next block of characters is reached by loading the pointer into the data base register.

Deletion of characters requires several steps. Unless the first character to be deleted is at the beginning of a four letter block and the last letter to be deleted is at the end, it is necessary to "void" a number of letters in the two end blocks. Then the pointer can be adjusted to bypass the remainder of the section to be removed. To insert a section, the characters to be added are packed four to a word and linked together in the form of a string. Unless the insertion is to occur after a letter that terminates a block on the main string, the block must be separated into two blocks with the end part placed at the end of the insertion string. Unused spaces are filled

with "void" characters.

Free storage can be maintained either by the use of a free storage string or a garbage collector. If the free string is used, a localized garbage collector should be used to minimize the number of "void" characters on strings.

Method 5 (Linked Linear Strings):

The most reasonable way to scan the linked linear string is to use the single character storage-to-storage instructions. The next character is accessed by incrementing the data base register or by loading a pointer into the data base register. The detection and handling of the pointer must be considered.

The addition and deletion of characters is complicated. If there are four or more characters to be deleted, a pointer is placed where the first characters were located. If there are fewer than four letters to be removed, the remaining letters are moved to a block obtained from free storage, and replaced by a pointer to their new location. A return pointer is then inserted after these new characters. The insertion process is slightly more involved. The characters to be added are strung out in a block obtained from free storage. The four characters from the main string following the point of insertion are moved to the end of the new block and replaced by a pointer. Special care must be taken to check the moved characters for the presence of a pointer.

The use of a garbage collector is the only way that free storage can be maintained. The efficiency of multi-programming is dependent upon the frequency and effectiveness of the garbage collector.

Method 6 (Linear String):

The characters on this string are trivially accessed by continually incrementing the data base register.

The insertion and deletion of characters is not difficult, but it is slow. The entire string can be recopied with the desired changes. Alternatively, to delete a section of the string, all characters to the right of the section to be deleted are moved left a number of places corresponding to the number of characters to be deleted. To add to the string, all the characters following the point of insertion are moved right the correct number of places and the new characters are inserted.

There is no need for any additional storage maintenance, since characters are always stored at 100 per cent efficiency. This method is probably the most effective for operating in a multiprogramming environment.

VII. Summary of Data Structures

No single method can be determined "best" or "worst". Each has advantages and disadvantages; it is the application that will usually determine the most desirable method. The following table summarizes the characteristics of the six methods proposed.

	Packing density	Ease of scan	Ease of insert delete	Speed of insert delete	Multi-programming
(1) Double-word	12.5	easy	easy	fast	poor
(2) Single-word	25	easy	easy	fast	poor
(3) Variable Length	12.5-25	moderate	moderate	moderate	fair
(4) Packed Double	12.5-50	moderate	difficult	slow	fair
(5) Linked Linear	25-100	moderate	difficult	very slow	good
(6) Linear	100	easy	moderate	very slow	excellent

Table 1. Data Structure Characteristics

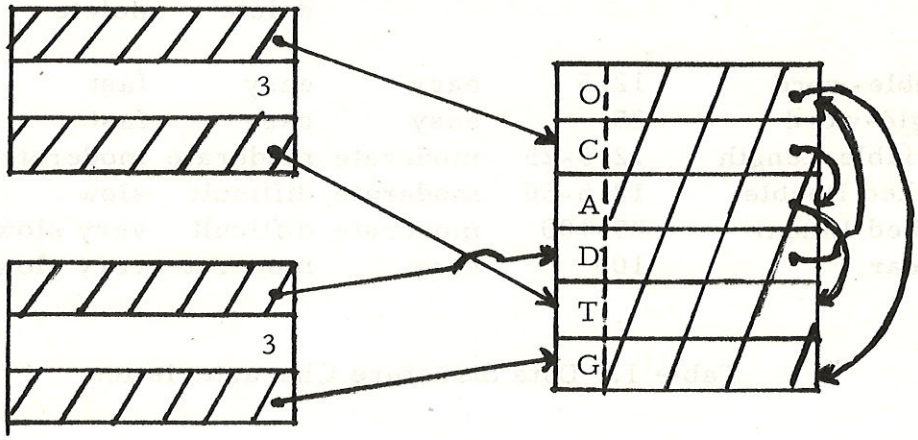
VIII. Example of String Processing on the System /360

The author decided to produce for the System/360 a string processing capability similar to that of COMIT and SNOBOL. In fact, the present system is SNOBOL compatible.

After considering the various data structures described in this paper, I chose Method 2, Single-word Blocks. Although packing density and application to multiprogramming were considered important, speed of operation and ease of implementation were given highest priority.

Strings are defined by a three word block called the "string reference block". The first word specified the location of the first character on the string, the second the length of the string, and the third, the location of the last character on the string. Although the string contents continually changed and were rearranged throughout memory, the string reference blocks remained at fixed locations and contained the information specifying

the present string contents. Figure 2. demonstrates this structure for the strings containing "CAT" and "DOG".



String Reference Blocks String Contents

Figure 2. String Reference Block Structure

A set of 30 basic string processing instructions is used. They are of the form: COPY Y, APPEND Y, REPLACE Y, INSERT Y, GOTO Y, etc. A program consists of a set of these instructions contained in the "program buffer". The program buffer is a section of memory containing consecutive 32 bits words; the first 8 bits of each word specify the instruction, the remaining 24 bits specify a string reference block in the case of a string manipulation, or the location of another instruction in the program buffer in the case of a GOTO. Figure 3 illustrates this structure.

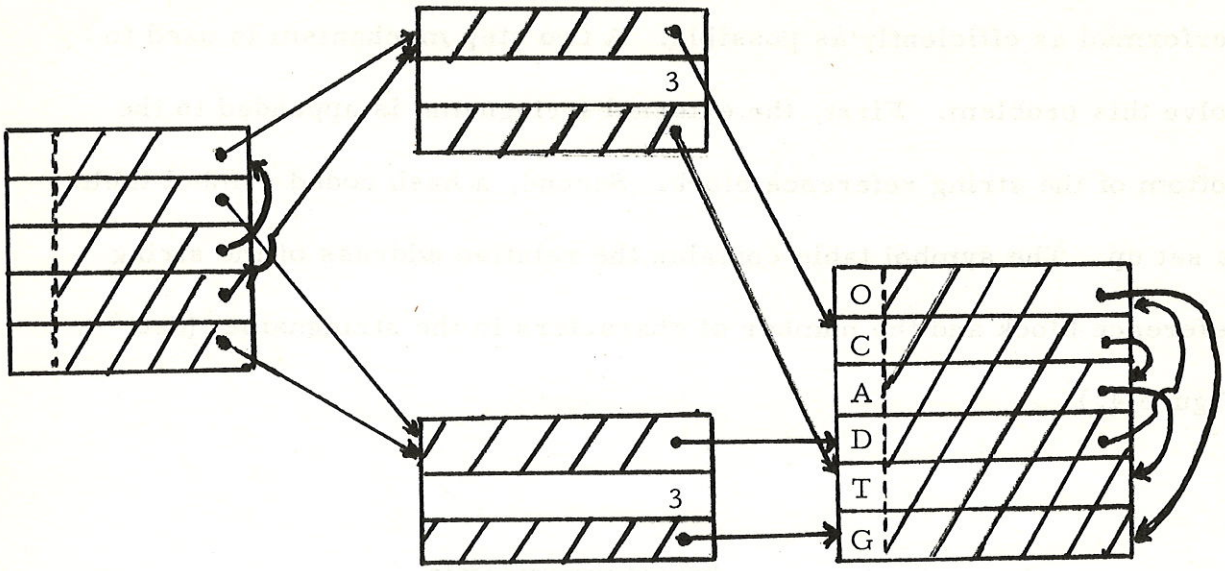


Figure 3. Program Buffer Structure

Strings that are to be variables are assigned external names, as is the case in most programming languages. The author decided to include the ability to indirectly reference a string. In indirect reference, rather than access a string by directly specifying its name (or reference block location), we specify the name of another string whose contents is the name of the string desired. Therefore, indirect string referencing requires a means by which the external stringnames (contained in a string) can be associated with the corresponding reference block location during execution. The problem of determining the reference block location of a string from its stringname is further complicated by the fact that string-names can be of arbitrary length, and can be created dynamically during

during execution. Of course, it is important that indirect referencing be performed as efficiently as possible. A two step mechanism is used to solve this problem. First, the external stringname is appended to the bottom of the string reference block. Second, a hash coded symbol table is set up. The symbol table contains the relative address of the string reference block and the number of characters in the stringname (See Figure 4.)

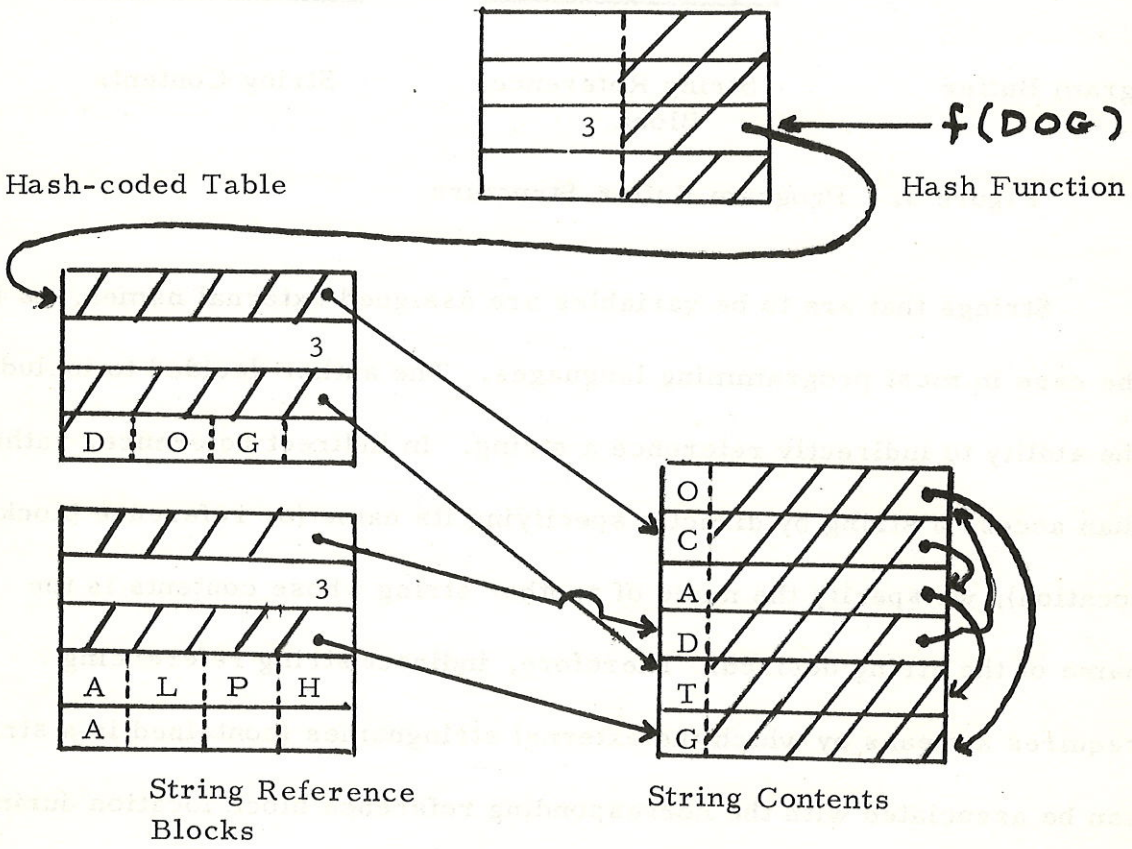


Figure 4. Overall Program-Data Structure

Referring to Figure 4, if we wanted to indirectly reference string ALPHA, the letters "DOG" (the contents of ALPHA) would be used as arguments of a hash function to determine the entry in the table. Since hash functions do not necessarily produce unique results, it is necessary to compare the stringname contained in the reference block indicated by the table entry with the letters "DOG". If the stringnames do not correspond, successive table entries are tried. In general, with a sufficiently large hash table the correct reference block can be located in one or two probes.

ACKNOWLEDGEMENTS:

Special thanks to Roy Harris, John Hershey, Larry-Stuart Deutsch, Payne Freret, Margaret Barovich, and Frank DeRemer for their assistance in preparing this paper.

REFERENCES:

(1) Arden, B.W., Galler, B. A., O'Brien, T.C., and Westervelt, F.H., "Program and Addressing Structure in a Time-Sharing Environment", J. ACM 13 (Jan. 1966)(1-17)

(2) Dennis, J.F., "Segmentation and the Design of Multiprogrammed Computer Systems", J. ACM 12, (Oct. 1965), (589-602).

(3) Farber, D. J., Griswold, R.E., and Polansky, I.P., "SNOBOL, A String Manipulation Language", J. ACM 11, (Jan. 1964) (21-30).

(4) Farber, D.J., et al., "SNOBOL 3", Bell Telephone Laboratories, Holmdel, N.J., (Unpublished).

(5) McCarthy, J. et al., LISP 1.5 Programmers Manual, M.I.T. Press, Cambridge, Mass. 1963

(6) McIlroy, M.D., "A String Manipulation System for FAP Programs", Bell Telephone Laboratories, Holmdel, N.J. (Unpublished).

(7) Madnick, Stuart E., "SPL/1: A String Processing Language", M.I.T. B.S.E.E. Thesis, June, 1966, Cambridge, Massachusetts.

(8) The M.I.T. Computation Center, The Compatible Time-Sharing System, A Programmers Guide, The M.I.T. Press, Cambridge, Massachusetts, 1963.

(9) Newell, A. Ed., Information Processing Language-V Manual, Prentice-Hall, 1961.

(10) Weizenbaum, J., "Symmetric List Processor", Comm. ACM 6, (Sept. 1963) (524-544).

(11) Yngve, V., COMIT Programmers Reference Manual, M.I.T. Press, Cambridge, Massachusetts, 1963.

IBM